**Ejercicios de Repaso**

**Actividad 1**

Crear una API REST en Node.js que gestione un CRUD de usuarios en una base de datos real, como **PostgreSQL**. Usaremos Postgre para almacenar los datos de los usuarios de manera persistente y simularemos el caso real de una pequeña aplicación de gestión de usuarios, por ejemplo, para una empresa que necesita registrar y administrar la información de sus empleados.

**Requisitos Previos**

1. **Node.js**: Asegúrate de tener Node.js instalado.
2. **PostgreSQL**: Asegúrate de tener PostgreSQL instalado y configurado. Crea una base de datos para este ejercicio (por ejemplo, usuarios\_db) y una tabla para los usuarios.
3. **Postman o Curl**: Para probar los endpoints.

**Paso 1: Crear la Base de Datos y la Tabla en PostgreSQL**

En PostgreSQL, crea una base de datos y una tabla de usuarios con el siguiente esquema:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

**Paso 2: Configurar el Proyecto e Instalar Dependencias**

Crea el proyecto:

Instala las dependencias:

* **express**: Para crear el servidor.
* **pg**: Para conectarse a PostgreSQL.
* **dotenv**: Para gestionar variables de entorno.

**Paso 3: Configurar el Entorno**

Crea un archivo .env en la raíz del proyecto con la configuración de conexión a PostgreSQL:

Imagen que contiene Patrón de fondo

Descripción generada automáticamente

**Paso 4: Configurar el Servidor y Rutas CRUD**

Crea un archivo server.js en la raíz del proyecto:

Configuración de conexión

Crear usuario

Obtener todos los usuarios

Actualizar un usuario por id

Eliminar un usuario por id

Iniciar el servidor

A tener en cuenta:

* **Conexión a PostgreSQL**: Utilizamos el objeto Pool del módulo **pg** para crear una conexión con PostgreSQL usando los datos de .env.
* **Consultas SQL**: En cada ruta CRUD, utilizamos consultas SQL para interactuar con la tabla usuarios:
  + **INSERT INTO** para crear.
  + **SELECT** para leer.
  + **UPDATE** para actualizar.
  + **DELETE** para eliminar.

**Paso 5: Probar la API en Postman o Curl**

Ahora que el servidor está configurado, puedes probar cada ruta en **Postman** o **Curl**.

**Crear un Usuario (POST)**

* **URL**: http://localhost:3000/usuarios
* **Método**: POST
* **Body**:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

**Obtener Todos los Usuarios (GET)**

* **URL**: http://localhost:3000/usuarios
* **Método**: GET

**Obtener un Usuario por ID (GET)**

* **URL**: http://localhost:3000/usuarios/{id}
* **Método**: GET

**Actualizar un Usuario por ID (PUT)**

* **URL**: http://localhost:3000/usuarios/{id}
* **Método**: PUT
* **Body**:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Eliminar un Usuario por ID (DELETE)**

* **URL**: http://localhost:3000/usuarios/{id}
* **Método**: DELETE

**Actividad 2**

Desarrollar una aplicación de chat básica en tiempo real donde los usuarios puedan enviar y recibir mensajes instantáneamente usando WebSockets.

**Instrucciones:**

1. **Configura el servidor Node.js**:
   * Crea un nuevo proyecto de Node.js.
   * Instala express para el servidor HTTP y ws para WebSockets.
   * Configura un servidor básico en Node.js que maneje las conexiones WebSocket.
2. **Desarrolla el Servidor WebSocket**:
   * Configura una conexión de WebSocket en el servidor.
   * Cuando un usuario se conecta, envía un mensaje de "bienvenida".
   * Escucha los mensajes de los clientes y retransmítelos a todos los usuarios conectados.
3. **Interfaz del Cliente**:
   * Crea una página HTML simple con un campo de entrada de texto y un botón de "Enviar".
   * Usa JavaScript en el lado del cliente para establecer una conexión WebSocket con el servidor.
   * Cuando un usuario envíe un mensaje, envíalo a través de WebSocket al servidor.
4. **Gestión de Mensajes**:
   * Al recibir un mensaje del servidor, el cliente debe mostrarlo en pantalla (simulando una conversación de chat).
   * Añade lógica para mostrar todos los mensajes en el área de chat y actualizarlos en tiempo real.
5. **Prueba la Aplicación**:
   * Abre varios clientes (por ejemplo, en varias pestañas o navegadores) y verifica que los mensajes enviados por un usuario se ven en los otros.
   * Realiza pruebas de conexión, desconexión y envíos múltiples de mensajes.

**Herramientas**:

* Usa Postman o cURL para probar conexiones WebSocket (también puedes observar el tráfico en el navegador).
* Puedes usar la extensión de Chrome "WebSocket Test Client" para verificar la comunicación en WebSocket.

**Extensión del Ejercicio**:

* Implementa nombres de usuario únicos.
* Añade notificaciones cuando un usuario se conecta o desconecta del chat.

**Actividad 3**

**1. Configuración de la Aplicación en LinkedIn**

Antes de comenzar a codificar, es necesario registrar tu aplicación en LinkedIn para obtener las credenciales necesarias.

1. Ve a [LinkedIn Developer Portal](https://www.linkedin.com/developers/) e inicia sesión.
2. Crea una nueva aplicación proporcionando la información requerida (nombre, logo, políticas, etc.).
3. En la configuración de la aplicación, obtén el **Client ID** y **Client Secret**, que usarás para la autenticación.
4. Configura los permisos (scopes) necesarios, como r\_liteprofile, r\_emailaddress o w\_member\_social dependiendo de la información que quieras obtener o publicar.

**2. Configuración del Proyecto Node.js**

**Paso 1: Crear un Proyecto de Node.js**

1. Crea una carpeta para tu proyecto y entra en ella:

**3. Estructura de Código**

**Paso 2: Configura las Variables de Entorno**

Crea un archivo .env para almacenar tus credenciales de LinkedIn:

Texto

Descripción generada automáticamente**Paso 3: Crear el Servidor Express**

Crea un archivo server.js para configurar el servidor y las rutas de autenticación:

**4. Explicación del Código**

* **/auth/linkedin**: Esta ruta redirige al usuario a la página de autenticación de LinkedIn, solicitando los permisos necesarios (r\_liteprofile para el perfil y r\_emailaddress para el correo electrónico).
* **/auth/linkedin/callback**: Después de que el usuario autoriza la aplicación, LinkedIn redirige al usuario a esta ruta, pasando un código de autorización en la URL.
  + El servidor usa este código para obtener un **token de acceso** de LinkedIn.
  + Con el **token de acceso**, se realiza una solicitud a la API de LinkedIn para obtener los datos del perfil y correo electrónico del usuario.

**5. Ejecución y Pruebas**

1. Inicia el servidor:
2. Visita http://localhost:3000/auth/linkedin en tu navegador para iniciar el flujo de autenticación de LinkedIn.
3. Después de autenticarse, el servidor redirigirá a /auth/linkedin/callback, donde verás los datos del perfil del usuario autenticado y su correo electrónico en formato JSON.

**Nota**: La API de LinkedIn está sujeta a restricciones de permisos, y puede requerir que tu aplicación sea revisada y aprobada por LinkedIn para obtener acceso a datos específicos.

**Actividad 4**

Imagina que estás desarrollando una aplicación de gestión de proyectos para una empresa. La aplicación permite que diferentes usuarios colaboren en proyectos, pero requiere que los usuarios se registren e inicien sesión para acceder a la información de los proyectos y realizar modificaciones. Solo los usuarios autenticados pueden acceder a los proyectos, y algunas rutas están restringidas a usuarios específicos.

**Requisitos**:

1. **Registro y autenticación:**
   * Los usuarios deben poder registrarse con un correo electrónico y una contraseña.
   * Los usuarios deben poder iniciar sesión con su correo y contraseña para obtener un token JWT.
2. **Acceso a proyectos:**
   * Los usuarios autenticados pueden obtener la lista de todos los proyectos.
   * Los usuarios autenticados pueden crear nuevos proyectos.
   * Los usuarios pueden ver detalles de un proyecto específico si están autenticados.
   * Solo los usuarios que han creado el proyecto pueden modificar o eliminar ese proyecto.
3. **Protección de rutas con JWT:**
   * Usar JWT para proteger las rutas de acceso a los proyectos.
   * Implementar middleware para verificar si el usuario está autenticado y autorizado para ciertas acciones.

**Instrucciones**

1. Configura el entorno en Node.js:
   * Configura un proyecto en Node.js con Express.
   * Instala las dependencias necesarias: express, jsonwebtoken, bcrypt (para encriptar contraseñas), y pg (para la base de datos PostgreSQL).
2. Crea las rutas de la API:
   * POST /register: para el registro de usuarios. Recibe email y password.
   * POST /login: para iniciar sesión. Devuelve un JWT si las credenciales son válidas.
   * POST /projects: para crear un proyecto. Requiere autenticación.
   * GET /projects: para obtener todos los proyectos. Requiere autenticación.
   * GET /projects/

: para obtener los detalles de un proyecto específico. Requiere autenticación.

* + PUT /projects/

y DELETE /projects/

: para modificar o eliminar un proyecto. Requiere autenticación y validación de permisos.

1. Implementa el middleware de autenticación:
   * Crea un middleware que verifique la validez del token JWT en cada solicitud a una ruta protegida.
2. Prueba el API:
   * Usa Postman o cualquier cliente REST para probar el flujo de registro, inicio de sesión y acceso a las rutas protegidas mediante el token.

**Ejemplo de flujo de uso:**

1. El usuario se registra en la aplicación (POST /register).
2. Luego, el usuario inicia sesión (POST /login) y obtiene un token.
3. El usuario usa el token para crear un proyecto (POST /projects).
4. El usuario puede ver todos los proyectos (GET /projects) y detalles de un proyecto específico (GET /projects/

).

1. Si el usuario es el creador del proyecto, puede modificar o eliminarlo (PUT/DELETE /projects/

**).**